

# Layer-specific parallelization for FPGA-based convolutional neural network accelerators: Performance and resource evaluation

Mustafa Tasci<sup>1</sup>, Ayhan Istanbulu<sup>2</sup>

Deep learning (DL) models require significant computational resources, making their deployment on edge devices with limited power and hardware capabilities challenging. Field-programmable gate arrays (FPGAs) provide an effective platform for accelerating such workloads because of their inherent parallelism and energy efficiency. This study investigates the impact of layer-wise parallelization levels, represented by folding coefficients, on the resource utilization and performance of FPGA-based DL accelerators, with a specific focus on convolutional (CONV) and fully connected (FC) layers. A LeNet-based accelerator model was implemented using the Xilinx FINN framework with W1A2 quantization (1-bit weights and 2-bit activations). Three folding coefficients, namely low (L), medium (M), and high (H), were defined for both the CONV and FC layers, yielding nine unique parallelization configurations. These accelerators were deployed on the PYNQ-Z1 platform and evaluated using the Fashion-MNIST dataset. A comprehensive evaluation quantifies key metrics, such as throughput (frames per second, FPS), resource utilization (look-up tables (LUTs), flip-flops (FFs), and block RAMs (BRAMs)), and power consumption. The results show that lower folding levels, corresponding to higher parallelism, significantly enhance the throughput, reaching up to 6809 FPS in the L-M and L-L configurations. This represents a 13-fold improvement over the baseline H-H configuration (C1) at the cost of increased resource usage. This study extends prior research on quantized neural networks (QNNs) by analyzing layer-specific parallelization strategies through adjustable folding factors and their effects on performance and resource trade-offs, offering valuable insights for optimizing FPGA-based deep learning (DL) inference in resource-constrained environments.

Keywords: convolutional neural network accelerators, field-programmable gate array, folding factors, parallelization

## 1 Introduction

The rapid proliferation of deep learning (DL) applications across domain, such as image recognition and autonomous system, has increased the demand for specialized hardware accelerators [1]. Traditional general-purpose processors, including central processing units (CPUs) and graphics processing units (GPUs), often face constraints in terms of power consumption and latency, particularly for real-time inference at the edge [2]. Field-programmable gate arrays (FPGAs) have emerged as promising alternatives, leveraging their inherent parallelism and energy efficiency to support deep neural network (DNN) deployment [3, 4]. The reconfigurable nature of FPGAs enables tailored hardware architectures that align with the computational graphs of DNNs, delivering significant performance enhancements and reduced power footprints compared with software-based solutions [5, 6].

DNNs, with their extensive layers and parameter counts, require efficient execution to achieve practical inference speeds [7]. Throughput, which reflects the overall processing capacity of neural network accele-

rators, is effectively quantified as frames per second (FPS), indicating the number of images processed per second on FPGA platforms, thereby addressing the demand for efficient inference. FPGA-based accelerators exploit various forms of parallelism, such as data, pipeline, and interlayer parallelism, to address this challenge [8]. Techniques such as fixed-point arithmetic, quantization, and pruning further reduce model complexity, enabling DNNs to operate within FPGA resource constraints while preserving acceptable accuracy [9]. Over the past decade, FPGA-based DL accelerators have rapidly developed. Initial designs featured handcrafted architectures optimized for specific convolutional neural networks (CNNs), such as NeuFlow for visual processing [10], and memory-centric implementations that aimed to reduce data transfer bottlenecks [11]. Recent research has focused on hardware optimization for CNN accelerators, highlighting efficient memory management, control logic, and computation strategies to improve their performance and resource utilization [12, 13]. Effective FPGA accelerator design hinges on strategic decisions regarding data movement, computational unit allocation,

<sup>1</sup> Department of Computer Technology, Bandırma Onyedi Eylül University, Balıkesir, Turkey

<sup>2</sup> Department of Computer Engineering, Balıkesir University, Balıkesir, Turkey  
[mtasci@bandirma.edu.tr](mailto:mtasci@bandirma.edu.tr)

and on-chip memory management to minimize the latency and maximize the throughput [14].

Quantization has become a pivotal technique for addressing these challenges, as it reduces the bit-width of weights and activations from 32-bit floating-point to lower representations (e.g., 8-bit, 4-bit, or binary), thereby decreasing the model size and computational complexity with minimal accuracy loss when appropriately implemented [15]. Quantized neural networks (QNNs), including extreme cases such as binary neural networks (BNNs), are central to efficient DL systems [16]. In FPGA deployment, quantization-aware strategies have been proposed, with BNNs simplifying multiplications to XNOR-based logic [17], as demonstrated in architectures such as FP-BNN [18]. Multibit quantization, such as W1A2 (1-bit weights and 2-bit activations), offers a compromise between hardware simplicity and inference accuracy [19]. Recent developments, such as FPQNet, showcase fully pipelined QNNs that achieve ultra-low latency and high throughput on FPGAs [20].

Folding, or resource sharing, is a critical hardware design technique for optimizing the FPGA implementation of CNNs. It maps multiple logical operations onto fewer physical processing units using temporal multiplexing [21]. In frameworks such as Xilinx FINN, folding is parameterized by processing elements (PEs) for output channel parallelism and single instruction multiple data (SIMD) for input channel parallelism, where lower PE and SIMD values indicate higher folding, reducing hardware resources (e.g., digital signal processors (DSPs), look-up tables (LUTs), and block RAMs (BRAMs)), but potentially increasing latency or decreasing throughput [22]. Conversely, unrolling – raising the PE and SIMD values – enhances parallelism, improving the throughput and reducing the latency at the cost of higher resource consumption [23]. Determining the optimal PE and SIMD configurations for different DNN layers remains a complex challenge, with the balance between resource utilization and performance directly impacting the feasibility of deploying large DNNs on resource-constrained FPGAs [24].

Building on our prior work on FPGA-based multi-layer perceptron (MLP) accelerators [25], this study investigates the impact of layer-specific parallelization on the performance, resource utilization, and power consumption of a W1A2-quantized LeNet model. The accelerator was implemented on a low-cost PYNQ-Z1 FPGA board using the Xilinx FINN framework and evaluated using the Fashion-MNIST dataset. In contrast to previous studies, which typically apply uniform folding configurations across all layers, this study systematically varies the folding coefficients (PE and SIMD) independently for the convolutional (CONV) and fully connected (FC) layers. This layer-specific parallelization strategy, realized through adjustable

folding in FINN, directly controls the degree of parallelism while trading off the throughput against resource utilization. The primary objectives are to quantify the distinct effects of folding on CONV versus FC layers, identify the optimal configuration that balances efficiency and performance, and provide practical insights for future FPGA-based deep learning accelerator designs.

## 2 Related work

The acceleration of DL workloads on FPGAs has emerged as a vibrant research domain, primarily driven by the demand for high-performance and energy-efficient inference on embedded and edge devices. CNNs, a dominant class of DL models, primarily consist of CONV and FC layers, each introducing distinct challenges and opportunities for hardware acceleration.

State-of-the-art FPGA-based CNN accelerators focus heavily on optimizing the computational cores of the CONV and FC layers [26]. For CONV layers, which involve numerous multiply accumulate (MAC) operations and exhibit locality and weight sharing, design techniques such as input tiling, filter unrolling, and pipelining are employed to maximize the spatial and temporal parallelism [27]. Efficient memory access patterns for weights and intermediate feature maps are critical for avoiding performance bottlenecks. In contrast, FC layers are characterized by dense matrix-vector multiplications and global connectivity, often leading to higher memory bandwidth demands and larger parameter counts [28]. Advanced implementations leverage high-bandwidth memory (HBM2) to alleviate data movement constraints in large-scale models [29]. A critical design choice for balancing resource utilization and latency across both layer types is the selection of folding factors, which dictate the degree of parallelism and resource reuse.

In parallel, the field has witnessed significant advancements in model compression, particularly quantization, to reduce the computational and memory footprint of neural networks. QNNs have emerged as a promising direction for hardware-efficient implementations, particularly in FPGAs. Binarized networks represent an extreme form of quantization and have been the focus of accelerators such as FCA-BNN [30] and FP-BNN [31], which utilize simple binary operations (e.g., XNOR, popcount) to achieve a throughput exceeding 1.9 tera operations per second (TOPS) on FPGA devices.

Beyond binarization, low-bit-quantized networks strike a balance between accuracy and efficiency. The Laius architecture, for example, supports fixed-point inference with custom bit widths tailored to application needs [32], whereas FPGA-NHAP provides a flexible platform supporting both spiking neural networks and

traditional QNNs [33]. These developments underscore the growing diversity of quantization strategies for FPGA-based DL deployment.

A particularly influential framework in this domain is FINN, which was introduced by Xilinx [34]. FINN adopts a dataflow-centric architecture that models QNNs as streaming pipelines with customizable folding mechanisms to balance the latency and resource usage. Its extension, FINN-R [35], enables automated design space exploration and supports a wide range of quantization strategies. These tools have significantly lowered the barrier for translating high-level DL models into efficient FPGA implementations.

Building on FINN's foundation, recent studies have expanded its applicability. For instance, entropy-based early exit mechanisms were integrated to reduce inference latency and power on constrained devices by terminating computations based on the prediction confidence [36]. An FPGA-based deep-learning Industrial Intrusion Detection System (I2DS) leveraging the FINN framework was developed for real-time anomaly detection in IoT environments [37], whereas scalable CNN deployments using FINN-R demonstrated the framework's versatility across domains [38].

The LeNet architecture, originally developed for digit classification, has been widely investigated in the context of FPGA acceleration. High-level synthesis (HLS)-based approaches have shown that LeNet can be efficiently implemented on low-cost FPGAs for real-time MNIST inference with low power consumption [39]. Other studies have investigated mixed-precision configurations by assigning different bit widths to layers to optimize the trade-off between accuracy and hardware cost [40]. A systematic evaluation of quantized LeNet variants under varying bit widths is presented in [41], highlighting how aggressive quantization impacts classification accuracy and resource utilization. Recently, FPQNet [20] pushed the boundaries by presenting a fully pipelined QNN accelerator for LeNet-like architectures that achieves ultra-low latency through aggressive quantization and pipelining strategies.

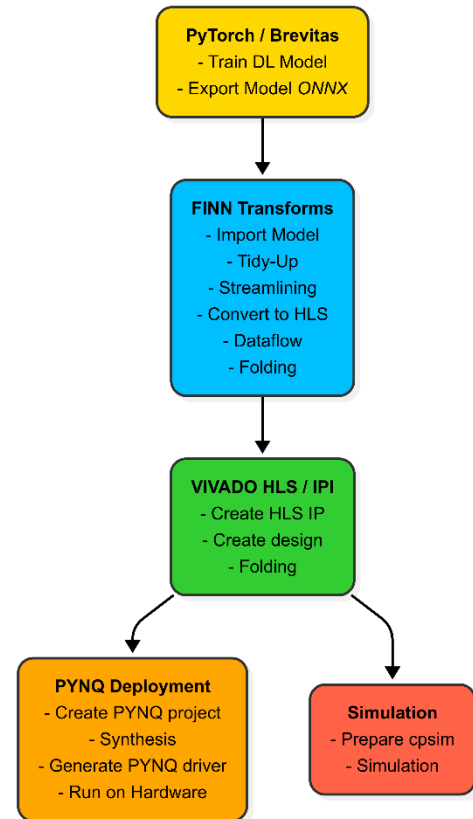
Mhaouch et al. demonstrated an efficient FPGA-based Conv2D layer accelerator on the PYNQ-Z2 platform, incorporating modules for input buffering, weight storage, bias addition, and fixed-point arithmetic supporting INT8, FP16, and FP32 formats. Optimized with loop unrolling, pipelining, and dataflow techniques, the INT8 configuration achieves a clock frequency of 174.74 MHz and power consumption of 1.7 W, making it suitable for real-time embedded medical imaging systems with reduced latency and energy consumption [42].

Despite these contributions, most studies have adopted fixed folding configurations or lack detailed layer-specific analyses. Few studies have systematically

explored quantized CNNs on low-cost platforms, such as the PYNQ-Z1, which is increasingly relevant for prototyping and education. This study differentiates itself by systematically varying the folding factors in a FINN-based quantized LeNet model with W1A2 quantization and conducting a comprehensive performance analysis across nine configurations using the Fashion-MNIST dataset on a PYNQ-Z1 board. This granular investigation addresses a literature gap and offers practical guidance for efficient QNN deployment on constrained FPGA platforms.

### 3 Materials and methods

This study systematically analyzed the impact of folding coefficients on the resource utilization and performance of an FPGA-based DL accelerator. The experimental setup utilized the Xilinx FINN framework to implement a quantized LeNet-based model. By leveraging tools such as PyTorch, Brevitas [43], and the FINN framework, the process ensures seamless integration from software modelling to hardware execution, allowing for a rigorous evaluation of architectural variations. The workflow is visually represented in Fig. 1, which captures the main phases and their interconnections, providing a high-level overview to guide the subsequent descriptions.



**Fig. 1.** Workflow for quantized CNN deployment using Xilinx FINN and PYNQ Z1

The diagram highlights five core stages: model creation in PyTorch/Brevitas, transformations within Xilinx FINN, synthesis via Vivado HLS/IPI, C++-based simulation for verification, and final deployment on PYNQ Z1 hardware. These phases form a pipeline that transitions from high-level neural network design to low-level hardware implementation, with each step contributing to the overall efficiency of the accelerator.

### 3.1 Dataset characteristics

In this study, the Fashion-MNIST dataset was used to evaluate the performance and accuracy of the proposed FPGA-based accelerators. Unlike the standard MNIST digit dataset, Fashion-MNIST consists of  $28 \times 28$  grayscale images of clothing items across 10 distinct categories (e.g., T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot). The dataset contained 70,000 images, partitioned into a training set of 60,000 images and a test set of 10,000 images. Each image is a single-channel (grayscale)  $28 \times 28$  pixel array, and the classes are uniformly distributed, with 7,000 images per category. The increased complexity of Fashion-MNIST compared to the basic MNIST dataset provides a more rigorous benchmark for the W1A2 quantized LeNet architecture.

### 3.2 System configuration and hyperparameters

To ensure the reproducibility of the experimental results and provide a clear overview of the implementation environment, hardware specifications, software frameworks, and training hyperparameters are consolidated in this section. The model training was performed using the Brevitas library within a Google Colab environment, and the resulting quantized models were deployed on the FPGA using the Xilinx FINN framework. The detailed system configurations are presented in Tab. 1.

The choice of W1A2 quantization (1-bit weights, 2-bit activations) in this study was made deliberately to isolate the effects of layer-specific parallelization. In our previous work on FPGA-based MLP accelerators [25], we extensively evaluated multiple quantization schemes, including W1A1, W1A2, and W2A2, across different network topologies. While these experiments focused on quantization accuracy and hardware efficiency trade-offs, the present study shifts the primary focus to layer-specific folding strategies.

W1A2 was selected because it offers an excellent compromise for the current analysis: binary weights enable highly efficient XNOR-popcount operations on FPGA, whereas 2-bit activations preserve sufficient representational capacity to maintain acceptable classification accuracy (93.4% on Fashion-MNIST). This

scheme is also natively and robustly supported by the FINN framework for streaming dataflow architectures, allowing us to vary the folding coefficients (PE and SIMD) independently for the CONV and FC layers without introducing additional variability from changing the bit widths. Thus, W1A2 serves as a fixed, well-characterized baseline that enables a clean and focused investigation of the performance–resource trade-offs arising solely from layer-specific parallelization in quantum neural networks.

**Table 1.** System configuration and hyperparameters

| Category | Parameter                | Specification      |
|----------|--------------------------|--------------------|
| Hardware | FPGA Board               | Xilinx PYNQ-Z1     |
|          | Logic Cells / DSP / BRAM | 85,000             |
|          | DSP                      | 220                |
|          | BRAM                     | 4.9MB              |
| Software | Training                 | Google Colab       |
|          | Quantization             | Brevitas           |
|          | Framework                | PyTorch            |
|          | Deployment               | Xilinx FINN        |
|          | Synthesis                | Vivado 2020.1      |
| Training | Dataset                  | Fashion-MNIST      |
|          | Optimizer                | SGD (Momentum=0.9) |
|          | Loss                     | Cross-Entropy      |
|          | Learning Rate            | 0.02               |
|          | Epochs                   | 100                |
|          | Quantization Level       | W1A2               |

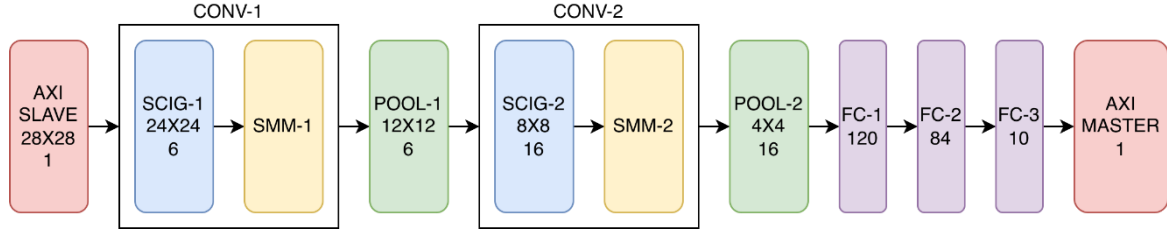
### 3.3 Model architecture

The core of our experimental setup is a LeNet-based accelerator model, which was chosen for its representative architecture featuring both CONV and FC layers, making it suitable for demonstrating layer-specific folding effects. As illustrated in Fig. 2, the model processes the input images through a sequence of layers in a dataflow manner optimized for FPGA implementation.

The pipeline begins with an advanced extensible interface (AXI) slave interface that receives  $28 \times 28 \times 1$  grayscale input images. The first CONV block (CONV-1) includes a sliding window unit (SCIG-1) generating  $24 \times 24 \times 6$  feature maps, followed by matrix-vector multiplication (SMM-1) for convolution operations, and a max-pooling layer (POOL-1) down sampling to  $12 \times 12 \times 6$ . This is succeeded by the second CONV block (CONV-2), where SCIG-2 produces  $8 \times 8 \times 16$  maps, SMM-2 performs the computations, and POOL-2 reduces to  $4 \times 4 \times 16$ . The extracted features are

then fed into a sequence of FC layers for final processing: first, a 120-neuron layer that flattens and integrates the spatial data, followed by an 84-neuron inter-

mediate layer. The process concludes with a 10-neuron output layer designed for classification, all interfaced via an AXI Master for efficient data transfer.



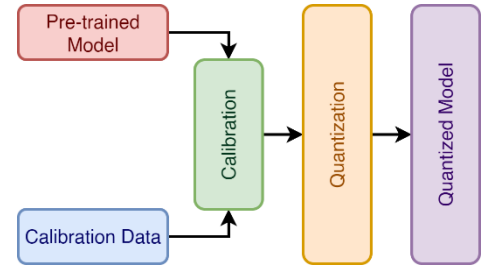
**Fig. 2** Hardware dataflow representation of the quantized LeNet-5 model, showing layer-wise processing from input to output

### 3.4 The Xilinx FINN framework

The Xilinx FINN framework is an open-source tool flow specifically designed for synthesizing highly optimized QNN inference accelerators on Xilinx FPGAs. FINN automates the process of mapping QNNs, particularly BNNs and ternary neural networks (TNNs), to highly parallel and pipelined hardware implementations. This is achieved by leveraging HLS to generate custom-tailored dataflow architectures. The framework emphasizes extreme quantization to maximize throughput and minimize resource consumption on FPGAs. FINN's modular and extensible design allows for the exploration of various hardware architectures and optimization parameters, including folding, for different network layers. Its applicability extends to various edge computing scenarios in which low-power, high-throughput inference is required, making it a pertinent tool for this study.

### 3.5 Quantization process

Quantization is a critical technique in DL for reducing model size and computational demands, making it particularly suitable for deployment on hardware, such as FPGAs. It involves mapping high-precision floating-point values to lower-bit integer representations, which can lead to faster inference and lower power consumption with a minimal loss in accuracy. The process typically includes calibration to determine the appropriate scaling factors, followed by the actual quantization step. Figure 3 illustrates the general workflow for quantization.



**Fig. 3.** Flow diagram: Quantization-aware training and layer-wise folding

This workflow starts from a pre-trained model and calibration data, proceeds through calibration to compute the necessary parameters, and culminates in the quantized model.

In uniform quantization, which is commonly used for its simplicity and efficiency, the following equations guide the process: For N-bit quantization, the maximum representable value (quantization threshold) is given by

$$Qnt_{th} = 2^{N-1} \quad (1)$$

This  $Qnt_{th}$  represents the upper limit of the quantized range. Next, for a matrix  $A$  to be quantized, the threshold  $th$  is the maximum absolute value in  $A$ :

$$th = \max |A_{ij}| \quad (2)$$

The scaling factor  $k$  is then computed as follows:

$$k = \frac{th}{Qnt_{th}} \quad (3)$$

Finally, the quantized matrix  $QntA$  is obtained by dividing  $A$  by  $k$  and clipping/rounding it to the integer range:

$$QntA = \frac{A}{k} \quad (4)$$

This scaled division ensures that the values fit within the quantized bit width, with clipping applied to handle any outliers and minimize the distortion.

While post-training quantization (PTQ) applies these steps after training using calibration data to estimate ranges, quantization-aware training (QAT) simulates quantization during training to better adapt the model to quantization. In this study, we adopted QAT to achieve superior accuracy under aggressive low-bit schemes, such as W1A2. For 1-bit weights (W1), the values are binarized to  $\pm 1$ , enabling XNOR and popcount operations that are highly efficient on FPGAs. Activations in 2-bit (A2) allow four discrete levels, balancing the precision and hardware simplicity.

To implement the QAT, we leveraged Brevitas, a PyTorch library that specializes in training QNNs. Brevitas provides flexible quantizers for weights and activations by inserting fake-quantization nodes during training to mimic the inference-time behavior. This allows the model to learn robust representations of quantization constraints.

The LeNet model was implemented by replacing standard PyTorch layers with their Brevitas quantized counterparts, which maps common PyTorch functions like `nn.Linear`, `nn.Conv2d`, and `nn.MaxPool2d` to Brevitas equivalents such as `QuantLinear`, `QuantConv2d`, and `QuantMaxPool2d`. This substitution enables precise control over the quantization parameters, including the bit widths for weights ( $w$ ), inputs ( $i$ ), biases ( $b$ ), and outputs ( $o$ ), which are specified via `QuantType` options such as `Int`, `Int8`, `Binary`, or `Ternary`. In this study, quantization was configured as W1A2, with 1-bit binary weights (`QuantWibol`,  $w=1$ , binary) for efficient XNOR operations and 2-bit activations to maintain a balance between precision and hardware efficiency. Separate quantizers were applied to the weights, activations, and inputs/outputs as needed, ensuring that the model adapts to the quantization noise during training.

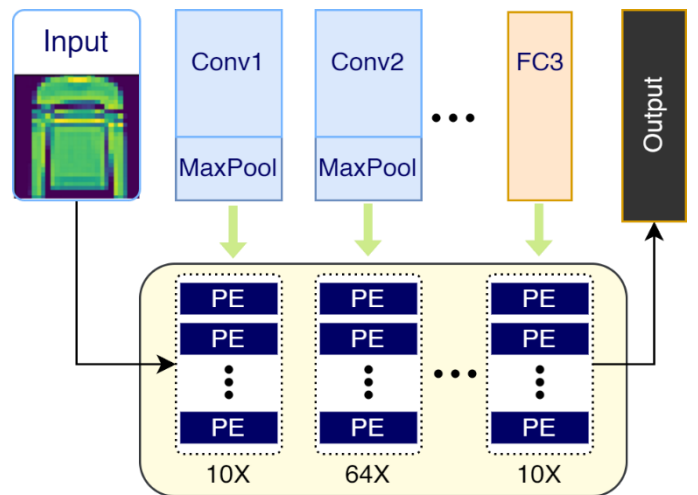
The model underwent 100 epochs of QAT during which the accuracy and loss were monitored. Following training, the model was exported to the Open Neural Network Exchange (ONNX) format using Brevitas's built-in ONNX optimizer, which preserves the quantization metadata and ensures compatibility with downstream tools, such as FINN, for hardware compilation. This export step, which was also performed on Google Colab, produced a portable representation ready for transformation and deployment.

### 3.6 Layer-specific parallelization via folding in FINN

Although the core focus of this study is layer-specific parallelization, its practical implementation within the Xilinx FINN framework was achieved through the folding mechanism. In FINN, folding serves as the key hardware optimization technique that controls the degree of parallelism by time-multiplexing the neural network operations onto a limited number of processing elements. Specifically, it governs the number of concurrent operations in nested loops (such as matrix-vector multiplications in the CONV and FC layers) through two configurable parameters: the number of processing elements (PE) for output-channel parallelism and the number of single-instruction multiple-data (SIMD) lanes for input-channel parallelism.

By independently tuning these folding coefficients (PE and SIMD) for the convolutional (CONV) and fully connected (FC) layers of the LeNet model, the design space can be explored granularly. This layer-specific approach enables a direct trade-off between the computational throughput and resource utilization while revealing the distinct parallelism requirements of each layer type. Figure 4 highlights the contrast between sequential CPU processing and parallel FPGA execution. On CPUs, operations are often bottlenecked by serial computations, as shown by the inhibited data flows. In FPGAs, multiple PEs enable concurrent processing across data streams, achieving significant speedups (e.g., 10 $\times$ , 84 $\times$ ) through distributed computation, which is fundamental to folding optimizations.

Two primary parameters govern the folding in FINN: the number of PEs (denoted as  $P$ ) and the number of SIMD lanes per PE (denoted as  $S$ ).



**Fig. 4.** Hardware acceleration of the LeNet model on FPGA, illustrating the mapping of convolutional and fully connected layers onto parallel processing elements (PEs)

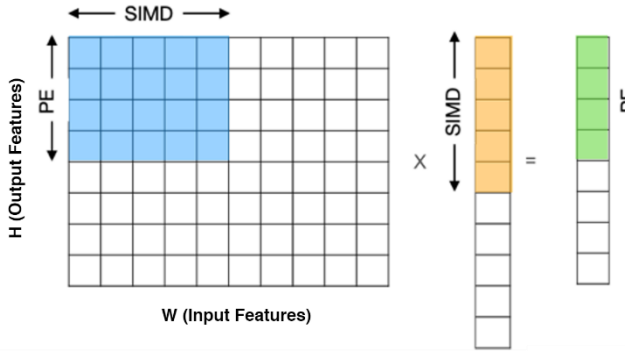
For a weight matrix of dimensions  $X \times Y$  (where  $X$  is the number of output features/neuron count and  $Y$  is the input features/synapse fan-in), the neuron fold  $F_n$  and synapse fold  $F_s$  are defined as

$$F_n = \frac{X}{P}, F_s = \frac{Y}{S} \quad (5)$$

The total fold  $F$  represents the number of clock cycles required to complete one matrix-vector multiplication.

$$F = F_n \times F_s \quad (6)$$

This tiling process is visualized in Fig. 5, where the weight matrix  $W$  is partitioned into a  $P$ -high and  $S$ -wide tile. The left grid represents the full matrix, with the height along the PE (output channels) and the width along the SIMD (input channels). The operation decomposes it into a SIMD-wide vertical strip (orange) multiplied by a PE-high horizontal strip (green), resulting in a processed output column.



**Fig. 5.** Illustration of matrix tiling for folding, showing how the PE and SIMD parameters divide the weight matrix  $W$  into computable segments

To ensure efficient mapping without padding,  $F_n$  and  $F_s$  must be integers. This requires that  $P$  exactly divides  $X$  and  $S$  exactly divides  $Y$ :

$$X \bmod P = 0, Y \bmod S = 0 \quad (7)$$

The values of  $P$  and  $S$  (and thus  $F_n$  and  $F_s$ ) are determined through design space exploration to achieve the user-specified throughput in frames per second (FPS), while respecting the resource constraints of the target FPGA.

Rate balancing ensures that the initiation interval (II) of each layer is properly aligned. The overall throughput is dictated by the maximum initiation interval across all layers ( $II_{\max}$ ):

$$\text{FPS} \approx \frac{f_{\text{clk}}}{II_{\max}} \quad (8)$$

where  $f_{\text{clk}}$  denotes the clock frequency. For fully-connected (FC) layers, the initiation interval equals the folding factor,  $II = F$ . Therefore, the required folding factor per layer is

$$F_n \cdot F_s = \frac{f_{\text{clk}}}{\text{FPS}} \quad (9)$$

If bottlenecks, such as limited memory channels or sliding window generation, occur, the initiation intervals of other layers are adjusted to match the throughput of the slowest layer. This approach prevents resource underutilization and hardware resource wastage.

Resource consumption increases with the level of parallelism and can be approximated as

$$R \approx k \cdot (S \cdot P \cdot b_w \cdot b_a) + R_{\text{base}} \quad (10)$$

where  $k$  is an operation-dependent constant,  $b_w$  and  $b_a$  represent the bit-widths of weights and activations, respectively, and  $R_{\text{base}}$  is the fixed resource overhead.

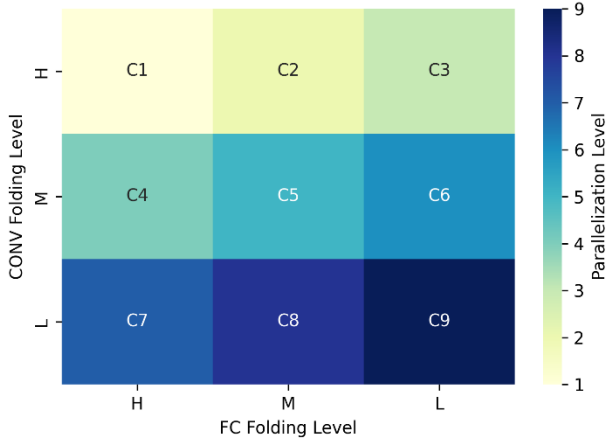
The design optimization problem is formulated as maximizing the throughput, subject to the available resources:

$$\max(\text{Throughput}) = \frac{f_{\text{clk}}}{F} \quad (11)$$

subject to  $R \leq R_{\text{avail}}$ .

This formulation allows for achieving a high FPS while ensuring that the design fits within the FPGA. Higher parallelism generally improves energy efficiency; however, it may also increase power consumption and lead to the underutilization of memory resources when non-unit folding factors are used.

By combining the three folding levels (L, M, and H) defined for the convolutional (CONV) layers with the three folding levels for the fully connected (FC) layers, nine unique folding configurations were systematically generated. The low (L), medium (M), and high (H) folding levels were empirically determined through preliminary experiments in FINN. These levels were chosen to remain within the resource limits of the PYNQ-Z1 board (particularly LUT and BRAM usage) while providing sufficiently distinct performance characteristics. The corresponding PE and SIMD values were selected to avoid exceeding the available hardware resources. The folding combinations are illustrated in Fig. 6.



**Fig. 6.** Matrix of folding configurations (C1-C9) for LeNet model. CONV folding levels (H, M, and L) are on the vertical axis and FC folding levels (H, M, and L) are on the horizontal axis.

**Low (L) folding** (higher parallelism/lower resource sharing): This configuration represents a higher degree of unrolling or parallelism, thereby increasing the demand for dedicated hardware resources. For CONV layers, this translates to more parallel processing elements for the filters or input channels. For FC layers, this implies a larger number of parallel MAC units. This setting is expected to yield higher performance but consume more resource.

**Medium (M) folding:** Representing an intermediate level of resource sharing, this configuration balances parallelism and resource utilization. It aims to strike a compromise between the throughput and hardware footprint.

**High (H) folding:** This configuration maximizes resource sharing, leading to a more serialized execution of operations on fewer processing elements. For CONV layers, this may imply fewer parallel filter operations. For FC layers, it would involve a smaller number of MAC units being reused more frequently. This setting is anticipated to minimize resource usage but results in lower performance.

Each configuration represents a distinct design point in the resource-performance trade-off space (Tab. 2).

**Table 2.** Folding configurations and corresponding parallelism parameters

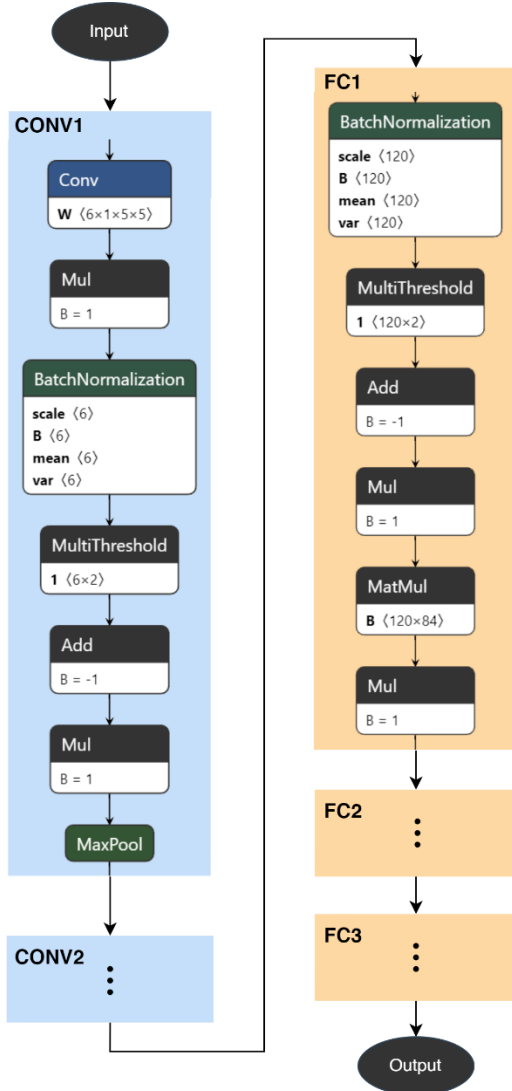
| Conf. | Param. | CONV1 | CONV2 | FC1     | FC2    | FC3   |
|-------|--------|-------|-------|---------|--------|-------|
|       |        | 1-6   | 6-16  | 256-120 | 120-84 | 84-10 |
| C1    | SIMD   | 1     | 1     | 1       | 1      | 1     |
|       | PE     | 1     | 1     | 1       | 1      | 1     |
| C2    | SIMD   | 1     | 1     | 8       | 8      | 1     |
|       | PE     | 1     | 1     | 8       | 4      | 5     |
| C3    | SIMD   | 1     | 1     | 8       | 60     | 1     |
|       | PE     | 1     | 1     | 60      | 42     | 5     |
| C4    | SIMD   | 1     | 3     | 1       | 1      | 1     |
|       | PE     | 3     | 8     | 1       | 1      | 1     |
| C5    | SIMD   | 1     | 3     | 8       | 8      | 1     |
|       | PE     | 3     | 8     | 8       | 4      | 5     |
| C6    | SIMD   | 1     | 3     | 8       | 60     | 1     |
|       | PE     | 3     | 8     | 60      | 42     | 5     |
| C7    | SIMD   | 1     | 6     | 1       | 1      | 1     |
|       | PE     | 6     | 16    | 1       | 1      | 1     |
| C8    | SIMD   | 1     | 6     | 8       | 8      | 1     |
|       | PE     | 6     | 16    | 8       | 4      | 5     |
| C9    | SIMD   | 1     | 6     | 8       | 60     | 1     |
|       | PE     | 6     | 16    | 60      | 42     | 5     |

\* The numbers in the header row (e.g., 1-6, 6-16, 256-120, etc.) indicate the input and output dimensions of the corresponding layers.

The specific numerical values for these **L**, **M**, and **H** settings were derived based on preliminary empirical exploration within the FINN framework, ensuring a distinct and measurable impact on the resource and performance metrics for the chosen LeNet architecture. For the L folding level, the PE and SIMD coefficients were experimentally selected to maximize the utilization of the Xilinx PYNQ-Z1 board's available hardware resources without exceeding them. These coefficients were configured according to the layer type (CONV or FC) and specific input/output dimensions of each layer, such as setting PE to match or scale with output channels in CONV layers and adjusting SIMD based on input channels, whereas similar tailoring was applied to FC layers for optimal parallelism. For each of these nine configurations, a complete LeNet-based accelerator was synthesized using the Xilinx FINN tool flow, targeting a representative FPGA device. The synthesis results provided detailed reports on resource consumption and enabled the performance characterization.

### 3.7 FINN compilation flow

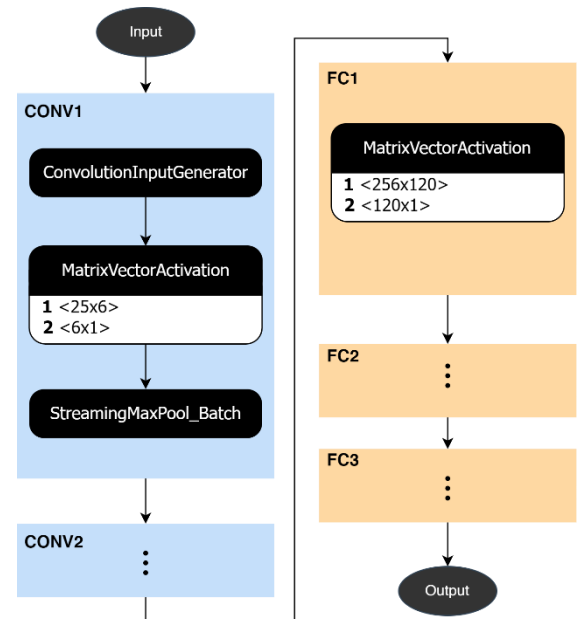
Following the training and export of the model to the ONNX format, the LeNet architecture was implemented as a QNN using the Brevitas library. This architecture consists of two CONV layers, each followed by max pooling operations, producing six and 16 output channels, respectively, and three FC layers with output dimensions of 120, 84, and 10 neurons. In the initial ONNX visualization, the blue blocks represent the CONV and pooling stages. Specifically, the first blue block corresponds to the 6-output-channel convolution and pooling, while the second handles the 16-output-channel equivalent, followed by yellow blocks denoting the sequential FC layers. Figure 7 shows blue blocks for the CONV and pooling stages, followed by yellow blocks for the FC layers.



**Fig. 7.** ONNX representation of the trained LeNet model prior to FINN transformations

Upon integration into the FINN framework, a series of transformations were systematically applied to optimize the model for hardware deployment: initially, TidyUp was used for general model cleanup and simplification, PrePost for handling pre- and post-processing elements, and Streamline for streamlining arithmetic operations. After the initial Streamline pass, the LowerConvsToMatMul transformation was employed to convert CONV nodes into Im2Col operations, which unfold input features into sliding-window arrays, thereby replacing the convolution with straightforward matrix multiplications. In the resulting bitstream model, each row of the input matrix is multiplied element-wise with the flattened kernel matrix, and the products are aggregated using parallel adders to generate feature maps.

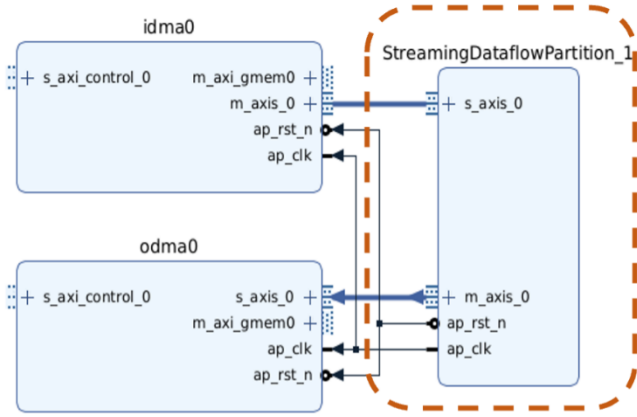
Given the 1-bit quantization parameter adopted in the network design, the ConvertBipolarMatMulToXnorPopcount transformation was subsequently applied, enabling the realization of matrix multiplications through XNOR logic gates and popcount counters for efficient binary operations. As a result of these transformations, the original CONV nodes (Conv) are observable as having been decomposed into Im2Col and MatMul nodes. A final streamline transformation was then reapplied, which rewrote the subtraction (Sub) nodes as additions, division (Div) nodes as multiplications, and batch normalization (BatchNorm) nodes as combinations of multiplications and additions. Furthermore, nodes such as BatchNorm, Mul, and Add were consolidated and replaced with multithreshold activations to further enhance computational efficiency. The transformed model after these optimizations is illustrated in Fig. 8.



**Fig. 8.** ONNX representation of the LeNet model after the initial Streamline transformation in the FINN

Following the Streamline transformations, the model nodes were converted to HLS layers using the Finnhlslib library. Depending on the quantization scheme, this process transformed the MatMul or Xnor-PopcountMatMul layers, along with the MultiThreshold layers, into MatrixVectorActivation layers to facilitate hardware-efficient implementations.

The FINN model wrapper was then employed to incorporate the PE and SIMD folding parameters into the opened nodes, thereby parallelizing the accelerator for enhanced performance. In the FINN hardware generation phase, analogous to the MLP accelerator, the ZynqBuild module was utilized, and platform and clock period information were passed as parameters to the model wrapper, enabling the synthesis of the accelerator hardware on the Xilinx Vivado platform. The LeNet model and resultant accelerator hardware are shown in Fig. 9.



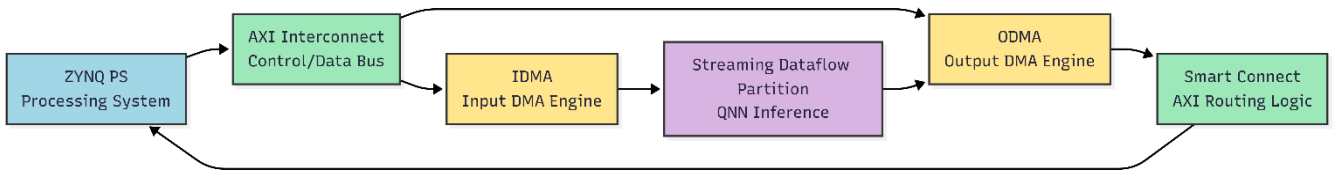
**Fig. 9** Vivado IP block diagram of LeNet model generated after FINN flow. The design consists of a streaming dataflow architecture with interconnected FINN IP cores between the input and output interfaces.

The central block in the Vivado IP diagram corresponds to the LeNet accelerator core. As shown in Fig. 10, the input DMA module (IDMA) is positioned above the accelerator core, and the output DMA module (ODMA) is located below it. These DMA modules are responsible for efficient data transfer between the LeNet accelerator and external DRAM. The accelerator is interfaced with the Zynq Processing System (PS) through AXI interfaces, utilizing the system-on-chip (SoC) architecture of the PYNQ board. The ARM processor streams the input data to the accelerator via pipelined DMA transfers and retrieves the processed outputs from the output DMA module.

### 3.8 Deployment and hardware evaluation

The deployment process involved transferring the synthesized bitstreams to the PYNQ-Z1 board via the secure shell (SSH) protocol, ensuring robust and secure integration with the target hardware platform. After deployment, a comprehensive hardware evaluation was conducted using a test set of 10,000 Fashion-MNIST images to assess the performance of the accelerator.

The key metrics included FPS to measure throughput, resource utilization extracted from Vivado synthesis and implementation reports (covering LUTs, LUTRAMs, FFs, and BRAMs), and power consumption evaluated through these methods. Power consumption was assessed using Vivado 2022 Power Tool estimates to analyze the dynamic and static power distributions across all configurations (C1-C9), with detailed breakdowns provided for specific cases (e.g., C3, C8, and C9, as shown in the subsequent results).

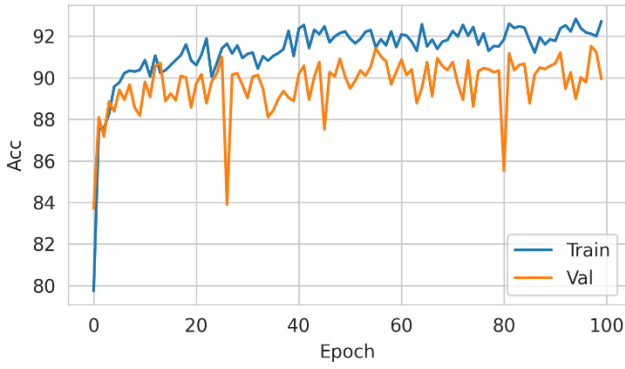


**Fig. 10.** Block diagram of the hardware architecture for a FINN-based QNN accelerator on a Zynq SoC

## 4 Results

This section presents the empirical results quantifying the impact of various folding configurations on the resource utilization and performance of the FPGA-based LeNet accelerator. A detailed analysis separates the effects on the CONV and FC layers, culminating in the identification of an optimal configuration.

Prior to the hardware-level experiments, the LeNet model was initially trained in floating-point precision on the Fashion-MNIST dataset using Google Colab for 100 epochs, achieving a test accuracy of 93.4%. Subsequently, the model was quantized with the W1A2 scheme using Brevitas, resulting in a post-quantization accuracy of 91.7% while maintaining a low and stable loss profile throughout training (see Fig. 11).



**Fig. 11.** Training accuracy and loss of the W1A2 LeNet model over 100 epochs

This minor accuracy drops from 93.4% to 91.7% demonstrates the effectiveness of W1A2 quantization in preserving model performance under aggressive low-bit constraints, closely aligning with the expected trade-off for such quantization levels. Although the classification performance remains satisfactory, the focus of this study is not solely on accuracy but rather on analyzing the interplay between parallelism, hardware resource usage, and inference performance. In subsequent stages, the weights of the trained and quantized model were deployed onto an FPGA platform for acceleration. The consistent training behaviour observed in the accuracy and loss curves ensured a reliable starting point for evaluating different folding configurations and their hardware-level implications.

The nine unique folding configurations, created by combining the L, M, and H folding coefficients for the CONV and FC layers, produced a broad range of resource usage and performance characteristics. The comprehensive dataset allows for a multidimensional analysis of the trade-offs inherent in hardware design for QNNs on FPGAs. Each configuration was synthesized and evaluated, providing quantitative insights into how the folding parameters directly influence the efficiency and speed of the accelerator.

Table 3 summarizes the resource utilization (LUT, FF, LUTRAM, BRAM) and performance metrics (throughput) for each of the nine folding configurations.

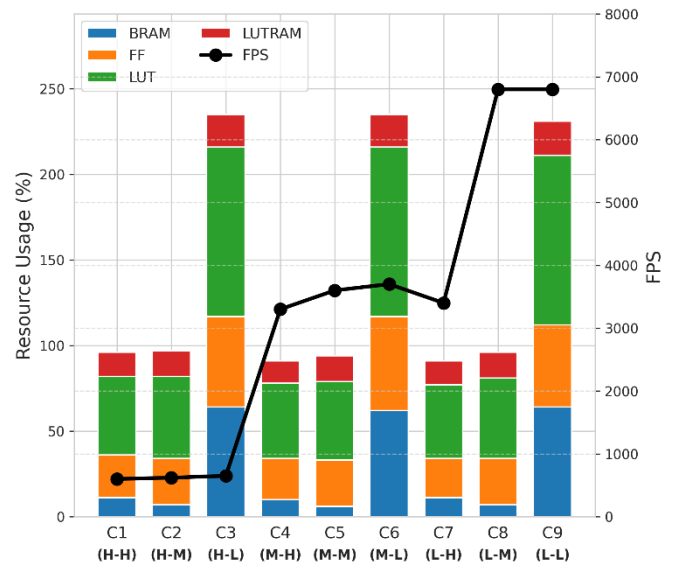
The table presents a comprehensive analysis of the resource utilization and performance metrics of various configurations of the LeNet accelerator implemented across different folding levels for the CONV and FC layers. The configurations, labelled C1 through C9, exhibit a range of folding combinations – H, M, and L – which significantly influence the throughput (FPS) and hardware resource consumption (LUT, LUTRAM, FF, and BRAM). Notably, configurations with L CONV folding (C7, C8, C9) demonstrated substantially higher throughput, peaking at 6809 FPS for C8 and C9,

compared to configurations with H or M CONV folding, where the throughput remained below 3500 FPS. This suggests that reducing the CONV folding enhances parallelism, thereby improving the inference speed.

**Table 3.** Resource utilization and performance of Lenet accelerators

| Conf. | Throughput (FPS) | LUT   | LUTRAM | FF    | BRAM |
|-------|------------------|-------|--------|-------|------|
| C1    | 535              | 12165 | 1183   | 13271 | 75   |
| C2    | 542              | 12677 | 1294   | 14779 | 50   |
| C3    | 535              | 26829 | 1669   | 29352 | 455  |
| C4    | 3253             | 11636 | 1149   | 12813 | 65   |
| C5    | 3471             | 12141 | 1262   | 14321 | 40   |
| C6    | 3471             | 26692 | 1677   | 29574 | 445  |
| C7    | 3253             | 11481 | 1197   | 12462 | 75   |
| C8    | 6809             | 12396 | 1351   | 14499 | 50   |
| C9    | 6809             | 26935 | 1767   | 26752 | 455  |

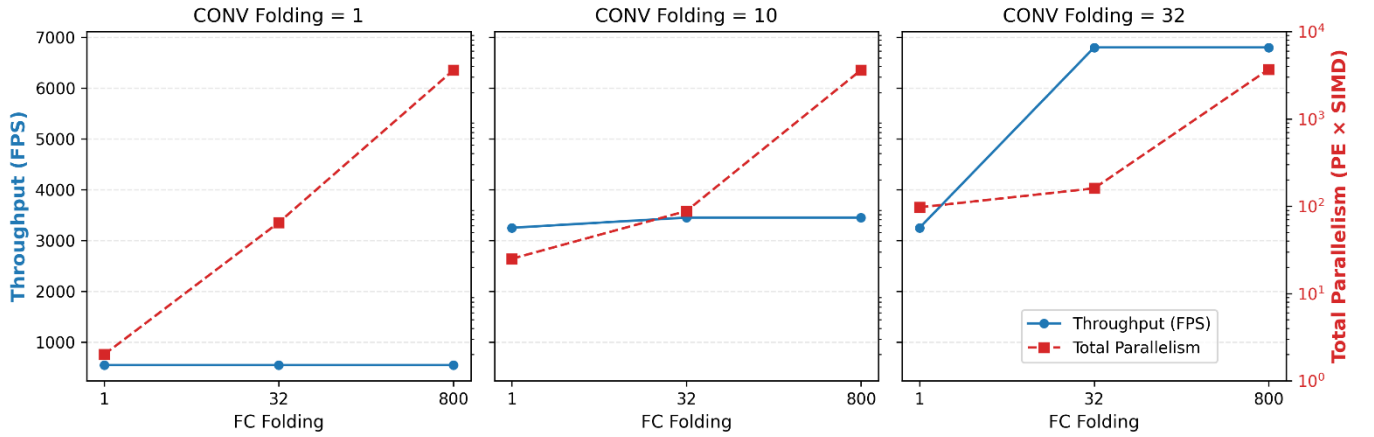
However, this performance gain is accompanied by increased resource utilization, as illustrated in Fig. 12, where configurations with L FC folding (e.g., C3: H-L, C6: M-L, C9: L-L) exhibit pronounced spikes in LUT (exceeding 50% in C3, C6, and C9) and BRAM (reaching approximately 32.5% in these cases), underscoring the hardware overhead associated with enhanced parallelism in FC layers.



**Fig. 12.** Resource utilization (LUT, LUTRAM, FF, BRAM) and throughput across different model configurations

Configurations with H CONV folding (Configurations 1-3: H-H, H-M, H-L) displayed the lowest throughput (approximately 536-542 FPS, as indicated by the flat line in the initial segment of the graph), coupled with moderate resource consumption, except for Model3, where L FC folding dramatically escalated LUT and BRAM demands, reflecting a disproportionate resource burden without commensurate performance benefits. In contrast, M CONV folding (Configurations 4-6: M-H, M-M, M-L) yields intermediate throughput

levels (3254–3471 FPS), with resource profiles varying according to FC folding. Notably, C6 shows a resource peak similar to C3 and C9, highlighting the sensitivity of the BRAM and LUT to the L FC configurations. The graph further emphasizes the efficiency of L CONV with M or L FC folding (Configurations 8-9: L-M, L-L), achieving the highest throughput (6809 FPS) with relatively balanced resource usage (LUT approximately 23-50%, BRAM 3.57-32.5%), suggesting an optimal throughput-to-resource ratio in these setups.



**Fig. 13.** Throughput variations across configurations based on layer-wise parallelism

Figure 13 provides a clear visualization of the inter-layer bottleneck in the LeNet accelerator across the nine configurations (C1 to C9), with throughput plotted as a blue line on the linear left y-axis and effective parallelism (computed as  $PE \times SIMD$ , averaged for CONV and FC layers) on the logarithmic right y-axis. The CONV parallelism remains constant within the folding groups – approximately 1 for C1-C3 (H CONV folding), 5.5 for C4-C6 (M), and 11 for C7-C9 (L) – exhibiting a strong positive correlation with throughput, which escalates from ~535-542 FPS in C1-C3 to ~3253-3471 FPS in C4-C6 and peaks at ~3253-6809 FPS in C7-C9. This stepwise increase underscores the computational dominance of the CONV layers, where enhanced parallelism (reflected in higher  $PE \times SIMD$  values for CONV1 and CONV2) directly mitigates the latency, as evidenced by the data. For instance, the transition from H to M CONV folding boosts the average CONV parallelism from 1 to 5.5, yielding a roughly six-fold throughput improvement.

Several critical observations regarding bottlenecks in the data flow across different configurations are particularly noteworthy.

- In configurations C2 and C3, the increased parallelism in the FC layers, compared with that in C1, significantly elevated the resource consumption without improving the throughput.
- Moderate parallelism in the CONV layers achieves a consistent intermediate throughput, independent of the FC layer parallelism. Notably, in configuration C4, moderate CONV parallelism results in a lower throughput than that achieved without FC parallelism, highlighting the dominance of the CONV bottleneck.
- In configuration C7, unlike in the other cases, the bottleneck shifted to the FC layer. CONV layers, configured with L folding for maximum parallelism, surpass the throughput capacity of the parallelized FC layer, thereby creating a bottleneck at the FC stage.
- In configurations C8 and C9, the maximum parallelism in the CONV layers, paired with M FC parallelism, effectively increased the throughput. However, in C9, maximizing FC parallelism further increases resource utilization without enhancing throughput because of a persistent bottleneck in the CONV layers.

Integrating Tab. 4, resource utilization patterns reinforce this bottleneck: configurations with elevated CONV parallelism (e.g., C7-C9) achieve superior throughput without proportional resource escalation (e.g., LUT ~11,461-12,346 for C7-C8 vs. ~26,935 for C9 with L FC), whereas L FC folding consistently

inflates BRAM and LUT (e.g., 45.5 BRAM in C3, C6, C9) without performance gains, indicating memory-bound constraints in FC that are overshadowed by CONV compute limitations.

Collectively, these observations affirm that the primary bottleneck resides in the CONV layers, where strategic reductions in folding (via optimized PE/SIMD) are essential for throughput enhancement, whereas FC adjustments primarily affect resource efficiency rather than overall speed in resource-constrained environments, such as the PYNQ-Z1 platform.

**Table 4.** Throughput, memory bandwidth, and latency analysis across various configurations

| Conf. | Throughput (FPS) | DRAM_in (KB/s) | DRAM_out (KB/s) | Latency (mS) |
|-------|------------------|----------------|-----------------|--------------|
| C1    | 535              | 420            | 0.53            | 1.869        |
| C2    | 542              | 440            | 0.74            | 1.845        |
| C3    | 535              | 420            | 0.53            | 1.869        |
| C4    | 3253             | 2550           | 3.25            | 0.307        |
| C5    | 3471             | 2720           | 3.47            | 0.288        |
| C6    | 3471             | 2720           | 3.47            | 0.288        |
| C7    | 3253             | 2550           | 3.25            | 0.307        |
| C8    | 6809             | 5340           | 6.80            | 0.147        |
| C9    | 6809             | 5340           | 6.80            | 0.147        |

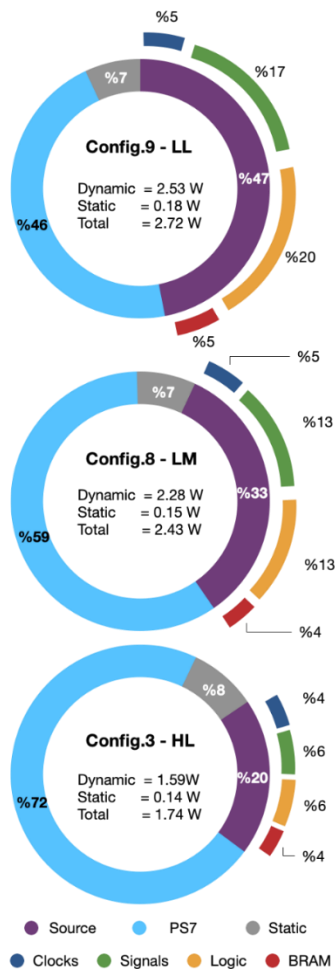
In the context of a LeNet-based accelerator implemented using the Xilinx FINN framework with W1A2 quantization (1-bit weights, 2-bit activations), the significant disparity between DRAM\_in and DRAM\_out, approximately three orders of magnitude, as observed in Tab. 4, arises from the combined effects of quantization, the LeNet architecture, and folding configurations defined by the PEs and SIMD parameters. Specifically, DRAM\_in, which represents the input data fetched from the DRAM (e.g., 28×28 grayscale images at 8-bit precision), exhibits high bandwidth requirements owing to the full-resolution data processed in the early CONV layers. Conversely, DRAM\_out, which comprises highly compressed 2-bit activation outputs, particularly in the final FC layers (e.g., 10-class outputs), results in substantially lower data volumes. The nine configurations (C1-C9) in Tab. 4 illustrate this, with DRAM\_in ranging from 420 to 5340 KB/s and DRAM\_out from 0.53 to 6.80 KB/s, reflecting the impact of varying PE and SIMD values across L, M,

and H folding factors. Configuration C8, which achieves the optimal performance-to-resource ratio with a throughput of 6809 FPS and a latency of 0.147 ms, likely balances PE and SIMD to maximize parallelism while efficiently managing DRAM access, highlighting the critical role of folding in minimizing DRAM\_out through quantization-driven data reduction while maintaining high throughput.

The analysis of latency, derived from throughput using the formula  $\text{latency} = 1 / \text{throughput}$  (FPS), provides compelling evidence of the dominant influence exerted by CONV folding levels on the overall inference time, with FC folding contributing minimally to this metric. Across the configurations examined, the latency values exhibited a distinct pattern: configurations with H CONV folding (C1-C3) demonstrated elevated latency ranging from approximately 1.845 to 1.869 mS, remaining consistent irrespective of variations in FC folding (H, M, L). In contrast, M CONV folding (C4-C6) reduced the latency to 0.288-0.307 mS, while L CONV folding (C7-C9) further decreased it to 0.147-0.307 mS, with the lowest values observed in configurations with M and L FC folding, although without notable differentiation.

The power consumption profiles for configurations C3, C8, and C9, as analyzed via Xilinx Vivado 2022 Power Tool (see Fig. 14), revealed that the static power and PS7 (Zynq processing system) consumption remained relatively stable, ranging from 0.148 W to 0.189 W (6.9%-8.4%) across configurations. In contrast, the source power, which is driven by logic resources and BRAM, varies significantly with the folding levels, increasing from 1.598 W (91.6%) in C3 to 2.534 W (93.1%) in C9. This escalation reflects the higher resource utilization with lower folding (e.g., C9 with L-L configuration), where logic and BRAM contributions rise from 20% and 4% in C3 to 20% and 5% in C9, respectively, owing to the enhanced parallelism from higher PE and SIMD settings.

The PYNQ-Z1 platform's power consumption, peaking at 2.723 W for C9, underscores its energy efficiency compared to alternative platforms. Typical GPU-based implementations (e.g., NVIDIA Jetson Nano) consume 5-10 W for similar DNN inference tasks, whereas low-power CPUs (e.g., Raspberry Pi) require 2-5 W. This positions the FPGA-based accelerator favorably, leveraging reconfigurability to achieve a high throughput of up to 6809 FPS at a fraction of the power draw, with source power variations highlighting the trade-off between parallelism and resource usage.



**Fig. 14.** Power consumption breakdowns for the LeNet accelerator: (a) configuration 3, (b) configuration 8, and (c) configuration 9, featuring elevated PS7 and resource usage, all analyzed via Xilinx Vivado 2022

## 5 Discussion and conclusions

The analysis of parallelism levels (PE and SIMD configurations) in FPGA-based DL accelerators yields important insights for hardware-design optimization. CONV layers exhibit greater tolerance to lower parallelism (fewer PEs and narrower SIMD lanes). This enables resource-efficient implementations with only mild performance degradation, particularly in the early layers with larger feature maps, thereby conserving LUT and BRAM resources. In contrast, FC layers are more sensitive to reduced parallelism and require higher PE counts and wider SIMD configurations to preserve an acceptable throughput and latency, particularly in the final classifier stage.

This study confirms the fundamental trade-off in FPGA accelerator design: lower parallelism reduces resource utilization (LUTs, FFs, and BRAMs) at the expense of increased latency and decreased throughput, whereas aggressive parallelism delivers higher FPS but rapidly exhausts on-chip resources. Among the nine

configurations, C8 (low folding for CONV layers and medium folding for FC layers) achieved the best performance-to-resource ratio with 6809 FPS, demonstrating that layer-specific parallelization is essential for balancing efficiency on resource-constrained platforms, such as the PYNQ-Z1.

The limitations of this study include the use of a relatively simple LeNet architecture with W1A2 quantization, which may limit generalizability to deeper networks (e.g., ResNet) or other quantization schemes (4-bit or 8-bit). Experiments were conducted exclusively with the Xilinx FINN framework on a specific FPGA family; the results may vary with different toolchains or vendors. Additionally, the empirically chosen “Low,” “Medium,” and “High” folding levels could benefit from a more exhaustive design space exploration.

In conclusion, this study demonstrates that parallelization must be applied in a layer-specific manner. The optimal configuration identified in this study was C8, which achieved the highest throughput of 6809 FPS while delivering the best performance-to-resource utilization ratio among all nine configurations. Although folding serves merely as the implementation mechanism in the FINN framework, the core contribution of this study lies in revealing that CONV layers benefit significantly from aggressive parallelization (low folding), whereas FC layers are more sensitive to resource overhead than to further increases in parallelism. These findings, clearly exemplified by the superior balance achieved in C8, provide concrete and actionable guidance for designing efficient FPGA-based deep learning accelerators tailored to the distinct computational characteristics of different neural network layers. Future work will extend this approach to more complex architectures, investigate dynamic PE/SIMD adaptation, and perform detailed power profiling across a wider range of quantization schemes.

## Acknowledgement

This work has been supported by Balıkesir University Scientific Research and Development Support Program (BAP) in Türkiye under project number 2020/091

## References

- [1] A. Shawahna, S. M. Sait, and A. El-Maleh, “FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review,” *IEEE Access*, vol. 7, pp. 7823–7859, 2019, doi: 10.1109/ACCESS.2018.2890150.
- [2] X. Liu et al., “Collaborative edge computing with FPGA-based CNN accelerators for energy-efficient and time-aware face tracking system,” *IEEE Trans. Comput. Soc. Syst.*, vol. 9, no. 1, pp. 252–266, 2021, doi: 10.1109/TCSS.2021.3059318.

- [3] R. Gadea-Gironés, J. Fe, and J. M. Monzo, "Task parallelism-based architectures on FPGA to optimize the energy efficiency of AI at the edge," *Microprocess. Microsyst.*, vol. 98, p. 104824, 2023, doi: 10.1016/j.micpro.2023.104824.
- [4] M. Tibaldi and C. Pilato, "A survey of FPGA optimization methods for data center energy efficiency," *IEEE Transactions on Sustainable Computing*, vol. 8, no. 3, pp. 343-362, 2023, doi: 10.1109/TSUSC.2023.3273852.
- [5] Y. Liang et al., "An efficient hardware design for accelerating sparse CNNs with NAS-based models," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 3, pp. 597-613, 2021, doi: 10.1109/TCAD.2021.3066563.
- [6] D. Ghimire, D. Kil, and S. Kim, "A survey on efficient convolutional neural networks and hardware acceleration," *Electronics (Basel)*, vol. 11, no. 6, p. 945, 2022, doi: 10.3390/electronics11060945.
- [7] X. Zhang et al., "DNNBuilder: An automated tool for building high-performance DNN hardware accelerators for FPGAs," in *Proceedings of the International Conference on Computer-Aided Design*, 2018, pp. 1-8. doi: 10.1145/3240765.3240801.
- [8] G. Brignone, R. Bosio, F. Ottati, C. Sansoè, and L. Lavagno, "SILVIA: Automated Superword-Level Parallelism Exploitation via HLS-Specific LLVM Passes for Compute-Intensive FPGA Accelerators," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 18, no. 2, pp. 1-16, 2025, doi: 10.1145/3705324.
- [9] B. A. Motetti, M. Risso, A. Burrello, E. Macii, M. Poncino, and D. J. Pagliari, "Joint pruning and channel-wise mixed-precision quantization for efficient deep neural networks," *IEEE Transactions on Computers*, 2024, doi: 10.1109/TC.2024.3449084.
- [10] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, "NeuFlow: A Runtime Reconfigurable Data-flow Processor for Vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2011, pp. 109-116. doi: 10.1109/CVPRW.2011.5981829.
- [11] M. Peemen, A. A. A. Setio, B. Mesman, and H. Corporaal, "Memory-Centric Accelerator Design for Convolutional Neural Networks," in *Proceedings of the IEEE International Conference on Computer Design*, 2013, pp. 13-19. doi: 10.1109/ICCD.2013.6657019.
- [12] T. Ma, Z. Li, Q. Li, H. Liu, Z. Zhao, and Y. Wang, "FPGA Optimized Accelerator of DCNN with Fast Data Readout and Multiplier Sharing Strategy," *Computers, Materials & Continua*, vol. 77, no. 3, pp. 3237-3263, 2023, doi: 10.32604/cmc.2023.045948.
- [13] V. Leon, S. Mouselinos, K. Koliogeorgi, S. Xydis, D. Soudris, and K. Pekmestzi, "A TensorFlow Extension Framework for Optimized Generation of Hardware CNN Inference Engines," *Technologies (Basel)*, vol. 8, no. 1, p. 6, 2020, doi: 10.3390/technologies8010006.
- [14] H. Hong et al., "Survey of convolutional neural network accelerators on field-programmable gate array platforms: architectures and optimization techniques," *J. Real. Time. Image Process.*, vol. 21, no. 3, p. 64, 2024, doi: 10.1007/s11554-024-01442-8.
- [15] U. Kulkarni et al. (2021). *Performance Improvements in Quantization Aware Training and Appreciation of Low Precision Computation in Deep Learning*. In: Thampi, S.M., Krishnan, S., Hegde, R.M., Ciunzo, D., Hanne, T., Kannan R., J. (eds) *Advances in Signal Processing and Intelligent Recognition Systems. SIRS 2020. Communications in Computer and Information Science*, vol 1365. Springer, Singapore. [https://doi.org/10.1007/978-981-16-0425-6\\_79](https://doi.org/10.1007/978-981-16-0425-6_79).
- [16] Y. Su, K. P. Seng, L. M. Ang, and J. Smith, "Binary Neural Networks in FPGAs: Architectures, Tool Flows and Hardware Comparisons," *Sensors*, vol. 23, no. 22, p. 9254, Nov. 2023, doi: 10.3390/s23229254.
- [17] C. Yuan and S. S. Agaian, "A comprehensive review of binary neural network," *Artif. Intell. Rev.*, vol. 56, no. 11, pp. 12949-13013, 2023, doi: 10.1007/s10462-023-10464-w.
- [18] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, "FP-BNN: Binarized Neural Network on FPGA," *Neurocomputing*, vol. 275, pp. 1072-1086, 2018, doi: 10.1016/j.neucom.2017.09.023.
- [19] Q. Cheng et al., "Reliability exploration of system-on-chip with multi-bit-width accelerator for multi-precision deep neural networks," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 70, no. 10, pp. 3978-3991, 2023, doi: 10.1109/TCSI.2023.3300899.
- [20] M. Ji, Z. Al-Ars, P. Hofstee, Y. Chang, and B. Zhang, "Fpqnnet: Fully pipelined and quantized cnn for ultra-low latency image classification on fpgas using opencapi," *Electronics (Basel)*, vol. 12, no. 19, p. 4085, 2023, doi: 10.3390/electronics12194085.
- [21] M. Amin and T. Adiono, "Area optimized CNN architecture using folding approach," in *2019 International Conference on Electrical Engineering and Informatics (ICEEI)*, 2019, pp. 206-209. doi: 10.1109/ICEEI47359.2019.8988879.
- [22] L. Liu et al., "An Automatic Neural Network Architecture-and-Quantization Joint Optimization Framework for Efficient Model Inference," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 5, pp. 1497-1510, 2023, doi: 10.1109/TCAD.2023.3339438.
- [23] R. Wu, X. Guo, J. Du, and J. Li, "Accelerating neural network inference on FPGA-based platforms – A survey," *Electronics (Basel)*, vol. 10, no. 9, p. 1025, 2021, doi: 10.3390/electronics10091025.
- [24] J.-Y. Kim, "FPGA based neural network accelerators," in *Advances in computers*, vol. 122, Elsevier, 2021, pp. 135-165. doi: 10.1016/b.s.adcom.2020.11.002.
- [25] M. Tasci, A. Istanbulu, V. Tumen, and S. Kosunalp, "FPGA-QNN: quantized neural network hardware acceleration on FPGAs," *Applied Sciences*, vol. 15, no. 2, p. 688, 2025, doi: 10.3390/app15020688.
- [26] D. T. Nguyen, H. Kim, and H.-J. Lee, "Layer-Specific Optimization for Mixed Data Flow With Mixed Precision in FPGA Design for CNN-Based Object Detectors," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 6, pp. 2450-2464, Jun. 2021, doi: 10.1109/tcsvt.2020.3020569.
- [27] J. Wang, W. Tong, and X. Zhi, "Model parallelism optimization for CNN FPGA accelerator," in *Algorithms*, vol. 6, no. 2 MDPI, 2023, p. 110. doi: 10.3390/a16020110.
- [28] K. He, B. Liu, Y. Zhang, A. Ling, and D. Gu, "FeCaffe: FPGA-enabled Caffe with OpenCL for Deep Learning Training and Inference on Intel Stratix 10," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, New York, NY, USA: Association for Computing Machinery, 2020, p. 314. doi: 10.1145/3373087.3375389.
- [29] S. K. Venkataramanaiah et al., "FPGA-based low-batch training accelerator for modern CNNs featuring high bandwidth memory," in *Proceedings of the 39th International Conference on Computer-Aided Design*, New York, NY, USA: Association for Computing Machinery, 2020. doi: 10.1145/3400302.3415643.
- [30] J. Gao, Y. Yao, Z. Li, and J. Lai, "Fca-bnn: Flexible and configurable accelerator for binarized neural networks on fpga," *IEICE Trans. Inf. Syst.*, vol. 104, no. 8, pp. 1367-1377, 2021, doi: 10.1587/transinf.2021EDP7054.
- [31] R. Zhao et al., "Accelerating Binarized Convolutional Neural Networks with Software-Programmable FPGAs," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 15-24. doi: 10.1145/3020078.3021741.

- [32] Z. Nie et al., "Laius: an energy-efficient FPGA CNN accelerator with the support of a fixed-point training framework," *International Journal of Computational Science and Engineering*, vol. 21, no. 3, pp. 418-428, 2020, doi: 10.1504/IJCSE.2020.106064.
- [33] Y. Liu, Y. Chen, W. Ye, and Y. Gui, "FPGA-NHAP: A General FPGA-Based Neuromorphic Hardware Acceleration Platform with High Speed and Low Power," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 6, pp. 2553-2566, 2022, doi: 10.1109/TCSI.2022.3160693.
- [34] Y. Umuroglu et al., "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 65-74. doi: 10.1145/3020078.3021744.
- [35] M. Blott et al., "FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 3, pp. 1-23, 2018, doi: 10.1145/3242897.
- [36] M. Kong and J. L. Nunez-Yanez, "Entropy-Based Early-Exit in a FPGA-Based Low-Precision Neural Network," in *International Symposium on Applied Reconfigurable Computing*, 2022, pp. 72-86, doi: 10.1007/978-3-031-19983-7\_6.
- [37] I. Morianos, K. Georgopoulos, A. Brokalakis, T. Kyriakakis, and S. Ioannidis, "I2DS: FPGA-Based Deep Learning Industrial Intrusion Detection System," in *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, 2024, pp. 165-176, doi: 10.1007/978-3-031-78380-7\_14.
- [38] T. Xaviour, B. Panuccio, E. Lee, L. Yang, and X. Wang, "Fast Implementation of Quantized Neural Network Accelerators on FPGAs for Image Classification," in *2025 Systems and Information Engineering Design Symposium (SIEDS)*, 2025, pp. 155-160. doi: 10.1109/SIEDS65500.2025.11021206.
- [39] E. Fatima, M. Fahad, H. Abrar, H. Waris, and others, "FPGA Based Artificial Neural Network Accelerator," in *2024 26th International Multi-Topic Conference (INMIC)*, 2024, pp. 1-6. doi: 10.1109/INMIC64792.2024.11004346.
- [40] S. Ben Ali, S.-I. Filip, O. Sentieys, and G. Lemieux, "MPTorch-FPGA: a Custom Mixed-Precision Framework for FPGA-based DNN Training," in *2025 Design, Automation & Test in Europe Conference (DATE)*, 2025, pp. 1-7. doi: 10.23919/DATE64628.2025.10993010.
- [41] B. J. Mohd, K. M. Ahmad Yousef, A. AlMajali, and T. Hayajneh, "Quantization-based optimization algorithm for hardware implementation of convolution neural networks," *Electronics (Basel)*, vol. 13, no. 9, p. 1727, 2024, doi: 10.3390/electronics13091727.
- [42] A. Mhaouch, W. Gtifa, T. Althobaiti, H. Faraj, and M. Machhout, "A Quality of Service Analysis of FPGA-Accelerated Conv2D Architectures for Brain Tumor Multi-Classification," *Computers, Materials & Continua*, vol. 84, no. 3, pp. 5637-5663, 2025, doi: 10.32604/cmc.2025.065525.
- [43] A. Pappalardo, A. Xilinx/Brevitas. 2021. Available online: <https://zenodo.org/records/16987789> (accessed on 10 April 2026).



**Mustafa Tasci** is an Assistant Professor at the Department of Computer Technology, Bandırma Onyedi Eylül University, Bandırma, Balıkesir, 10200, Turkey. His research interests include embedded systems, robotics and artificial intelligence. Tasci received his Ph.D. in electronics engineering from Balıkesir University



**Ayhan Istanbulu** is a Professor in the Department of Computer Engineering at Balıkesir University, Balıkesir, 10145, Turkey. His research interests include medical electronics, artificial intelligence and digital systems. Istanbulu received his Ph.D. in electronic computer education from Gazi University.

Received 1 March 2026